SAND REPORT

SAND2002-3790
Unlimited Release
Printed November 2002

Xyce™ Parallel Electronic Simulator

Users' Guide, Version 2.0

Scott A. Hutchinson, Eric R. Keiter, Robert J. Hoekstra, Lon J. Waters, Thomas V. Russo, Eric L. Rankin, Roger P. Pawlowski and Steven D. Wix

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/ordering.htm

SAND2002-3790
Unlimited Release
Printed November 2002

Xyce™ Parallel Electronic Simulator

Users' Guide, Version 2.0

Scott A. Hutchinson, Eric R. Keiter, Robert J. Hoekstra, Eric L. Rankin, Roger P. Pawlowski
Computational Sciences

Lon J. Waters, Thomas V. Russo and Steven D. Wix
Component Information and Models

January 9, 2004

Abstract

This manual describes the use of the **Xyce** Parallel Electronic Simulator code for simulating electrical circuits at a variety of abstraction levels. The **Xyce** Parallel Electronic Simulator has been written to support the simulation needs of the Sandia National Laboratories electrical designers in a rigorous manner. As such, the development has focused on improving the capability over the current state-of-the-art in the following areas:

- Capability to solve extremely large circuit problems by supporting large-scale parallel computing platforms (up to thousands of processors). Note that this includes support for most popular parallel and serial computers.
- Improved performance for all numerical kernels (e.g., time integrator, nonlinear and linear solvers) through state-of-the-art algorithms and novel techniques.
- A client-server or multi-tiered operating model wherein the numerical kernel can operate independently of the graphical user interface (GUI).
- Object-oriented code design and implementation using modern coding practices that ensure that the **Xyce** Parallel Electronic Simulator will be maintainable and extensible far into the future.

The code is a parallel code in the most general sense of the phrase - a message passing parallel implementation - which allows it to run efficiently on the widest possible

number of computing platforms. These include serial, shared-memory and distributed-memory parallel as well as heterogeneous platforms. Furthermore, careful attention has been paid to the specific nature of circuit-simulation problems to ensure that optimal parallel efficiency is achieved even as the number of processors grows.

Another feature required by designers is the ability to add device models, many specific to the needs of Sandia, to the code. To this end, the device package in the **Xyce** Parallel Electronic Simulator is designed to support a variety of device model inputs. These input formats include standard analytical models, behavioral models and look-up tables. Combined with this flexible interface is an architectural design that greatly simplifies the addition of circuit models.

One of the most important contribution **Xyce** makes to the designers at Sandia National Laboratories is in providing a platform for computational research and development aimed specifically at the needs of the Laboratory. With **Xyce**, Sandia now has an "in-house" capability with which both new electrical (e.g., device model development) and algorithmic (e.g., faster time-integration methods) research and development can be performed. Furthermore, these capabilities will then be migrated to the end users.

Acknowledgements

The authors would like to acknowledge the entire Sandia National Laboratories HPEMS (High Performance Electrical Modeling and Simulation) team, including Carolyn Bogdan, Regina Schells, Ken Marx, Steve Brandon, David Shirley and Bill Ballard, for their support on this project. We also appreciate very much the work of Becky Arnold and Mike Williamson for the help in reviewing this document.

Lastly, a very special thanks to Hue Lai for his help in typesetting this document in LaTeX.

Trademarks

Contacts

Bug Reports	http://tvrusso.sandia.gov/bugzilla
Email	xyce-support@sandia.gov
World Wide Web	http://www.cs.sandia.gov/Xyce

Contents

Figures

Tables

1. Preliminaries

Welcome to **Xyce**

The **Xyce** Parallel Electronic Simulator has been written to support, in a rigorous manner, the simulation needs of the Sandia National Laboratories electrical designers. It is targeted specifically to run on large-scale parallel computing platforms but also runs well on a variety of architectures including single processor workstations. It also aims to support a variety of devices and models specific to Sandia needs.

1.1 Xyce Overview

The **Xyce** Parallel Electronic Simulator development has focused on improving the capability over the current state-of-the-art in the following areas:

- Capability to solve extremely large circuit problems by supporting large-scale parallel computing platforms (up to thousands of processors). Note that this includes support for most popular parallel and serial computers.
- Improved performance for all numerical kernels (e.g., time integrator, nonlinear and linear solvers) through state-of-the-art algorithms and novel techniques.
- Support for modeling circuit phenomena at a variety of abstraction levels (device, analog, digital and mixed-signal) in a rigorous and tightly coupled manner, allowing for timely, full-system solutions.
- A client-server or multi-tiered operating model wherein the numerical kernel can operate distinctly from the simulation interface and the graphical user interface (GUI) (under development). This includes support for coupling with other simulation codes which may provide, for example, environmental information pertinent to the circuit (e.g, temperature as a function of time and space).
- Object-oriented code design and implementation using modern coding-practices that ensure that the **Xyce** Parallel Electronic Simulator will be maintainable and extensible far into the future.

The code is a parallel code in the most general sense of the phrase - a message passing parallel implementation - which allows it to run efficiently on the widest possible number of computing platforms. These include serial, shared-memory and distributed-memory parallel as well as heterogeneous platforms. Furthermore, careful attention has been paid to the specific nature of circuit-simulation problems to ensure that optimal parallel efficiency is achieved even as the number of processors grows.

As mentioned above, the **Xyce** Parallel Electronic Simulator is being developed in support of electrical designers of Sandia National Laboratory and, as such, is implementing several novel features that will make their job considerably easier. In addition to allowing the simulation of circuits of unprecedented size, **Xyce** includes novel approaches to numerical kernels such as time-stepping algorithms, nonlinear and linear solvers and improved device models. This approach aims to minimize the amount of simulation "tuning" required on the part of the designer and facilitate the code's successful usage.

Another feature of **Xyce** is the ability to add device models, many specific to the needs of Sandia, to the code. To this end, the device package in the **Xyce** Parallel Electronic Simulator is designed to support a variety of device model inputs. These input formats include standard analytical models, behavioral models and look-up tables, and support for

conductance values extracted from device-scale PDE models. Combined with this flexible interface is an architectural design that greatly simplifies the addition of circuit models.

For the user, this document contains a description of the **Xyce** Parallel Electronic Simulator, in which the following topics are specifically addressed. Chapters 2 and 3 give a simulation overview illustrating the distinctive features of **Xyce** and some examples of using the code. Chapters 4 through 5 describe how to create a basic circuit simulation design for **Xyce** using the standard netlist approach. Chapters 6 through 10 cover the creation and execution of circuit problems for **Xyce** on the supported platforms, including both serial and parallel architectures. This is followed by Chapter 11 that covers analyzing the results output by the code.

A companion document, the **Xyce** Reference Guide [1], contains more detailed information about a number of topics. Included in this document is a netlist reference for the input-file commands and elements supported within **Xyce**; a command line reference, which describes the available command line arguements for **Xyce**; and quick-references for users of other circuit codes, such as Orcad's PSpice [2] and Sandia's ChileSPICE.

1.2 Installation

To obtain a copy of **Xyce**, contact the **Xyce** development team at `http://www.cs.sandia.gov/Xyce`. Once you have the distribution file, install **Xyce** from the command line by following the instructions below. Examples are given for reference.

Instructions	Examples
Installation packages are named according the target operating system and architecture (parallel or serial).	`Install_Xyce_linux.tar.gz` (Linux Serial) `Install_Xyce_linux_MPI.tar.gz` (Linux Parallel) `Install_Xyce_windows.zip` (Windows)
Unpack the appropriate package for your platform. A similarly named installation directory is then created. Windows users can unpack with programs such as *WinZip*, *PKZip*, *Winrar*, etc.	`$ gzip -d Install_Xyce_linux.tar.gz` `$ tar xf Install_Xyce_linux.tar`
Enter this directory and run the installation shell script. Windows users should run the `install.bat` batch file.	`$ cd Install_Xyce_linux` `$ sh install_linux.sh`
Provide the requested information.	`Where should Xyce be installed?` `/usr/local/Xyce-2.0`

Completing the steps above will unpack **Xyce** to the specified directory. **IMPORTANT NOTE: if installing *both* serial and parallel versions of Xyce, you must specify different directories for each installation location. Failure to use different directories will cause the second installation to overwrite parts of the first and will likely yield an install that does not function.** Under the specified installation directories, the following subdirectories will be created:

- **bin** contains the executable used to start **Xyce**. The executable name will vary depending on the target operating system and architecture.
  - `runxyce` is the shell script for starting serial **Xyce** on Unix platforms.
  - `runxyce.bat` is the batch file for starting serial **Xyce** on Windows.
  - `xmpirun` is the wrapper script for mpirun used for running **Xyce** in parallel mode.
- **doc** contains the **Xyce** Users' Guide, comprehensive Reference Guide, and Release Notes. Read these for more information about this release and for detailed instructions on how to use **Xyce**.
- **lib** contains configuration files, libraries, and metadata for **Xyce**.
- **test** contains sample netlists and verification tools.

1.3 Quick Reference for Users of Other Circuit Codes

Xyce is targeted at Sandia's designer community, many of whom have experience using other circuit codes, such as Orcad PSpice and Sandia's ChileSPICE. Much of the use of **Xyce** mirrors closely that in other circuit codes, but there are occasional differences that are documented in the appendices. Users of PSpice and ChileSPICE can get a "quick-start" by looking at the **Xyce** Reference Guide. [1].

1.4 How to Use this Guide

This guide is designed so you can quickly find the information you need to use **Xyce**. It assumes that you are familiar with basic Unix-type commands, how Unix manages applications and files to perform routine tasks (e.g., starting applications, opening files and saving your work).

Typographical conventions

Before continuing in this Users' Guide, it is important to understand the terms and typographical conventions used. Procedures for performing an operation are generally numbered with the following typographical conventions.

Notation	Example	Description
`Verbatim text`	`mpirun -np 2 Xyce`	Commands entered from the keyboard on the command line or text entered in a netlist.
Bold Roman Font	Set nominal temperature using the **TNOM** option.	SPICE-type parameters used in models, etc.
Gray Shaded Text	DEBUGLEVEL	Feature that is designed primarily for use by **Xyce** developers.
`[text in brackets]`	`Xyce [options] <netlist>`	Optional parameters.
`<text in angle brackets>`	`Xyce [options] <netlist>`	Parameters to be inserted by the user.
`<object with asterisk>*`	`K1 <ind. 1> [<ind. n>*]`	Parameter that may be multiply specified.
`<TEXT1|TEXT2>`	`.PRINT TRAN` `+ DELIMITER=<TAB|COMMA>`	Parameters that may only take specified values.

Table 1.1. Xyce typographical conventions.

2. Distinctive Features of **Xyce**

Chapter Overview

This chapter outlines the disctinctive features of **Xyce** that provide the user with leading edge capability in a variety of circuit simulation areas.

- Section 2.1, **Xyce** *Capabilities*
- Section 2.2, *Support for large-scale parallel computing*
- Section 2.3, *Analysis Support within* **Xyce**

2.1 **Xyce** Capabilities

The **Xyce** Parallel Electronic Simulator is a program that models circuit behavior at a variety of levels of fidelity - from Partial Differential Equations (PDE) based device models to analog to mixed signal and digital simulation. When used with the simulation front-end, **Xyce** will serve as an electronic laboratory for circuit design and optimization before any hardware is ever implemented. There are many tools available for basic circuit design. What sets **Xyce** apart are several key improvements over the state-of-the-art in circuit simulation as described in this chapter.

2.2 Support for large-scale parallel computing

Xyce is a truly parallel simulation code, designed and written from the ground up to support large-scale (up to thousands of processors) parallel computing architectures. This gives **Xyce** the capability to solve circuit problems of unprecedented size in time frames that make these simulations practical.

Xyce is a parallel code that uses a message passing parallel implementation, which allows it to run efficiently on the widest possible number of computing platforms. These include serial, shared-memory and distributed-memory parallel as well as heterogeneous platforms. Furthermore, careful attention has been paid to the specific nature of circuit-simulation problems to ensure that optimal parallel efficiency is achieved even as the number of processors grows (*parallel scaling*).

Improved performance for all numerical kernels

In writing **Xyce** from scratch, new algorithms and heuristics have been used which improve the overall performance of the numerical kernels for a given level of accuracy. As an example, several new nonlinear solution options are available which, when coupled with iterative linear solvers, can reduce execution time for many problems.

Ease of device model addition

Another feature of **Xyce** is the ability to add device models, many specific to the needs of Sandia, to the code. To this end, the device package[1] in **Xyce** has been designed with a flexible device-model interface that greatly simplifies the addition of circuit models. The device package interface has support for the standard "analog" or SPICE-type models as well as look-up tables, behavioral models and even PDE-based device models.

[1]The term "package" is a Unified Modeling Language (UML) term which refers to a group of classes, something akin to a module and is largely used in object-oriented programming.

Problem-specific modeling fidelity

Xyce has been designed to support the needs of Sandia National Laboratories' electrical designers who often need to model circuit phenomena at varying levels of fidelity - even within a given simulation. Thus, the code has been designed with an infrastructure that supports coupled simulation at several distinct abstraction levels: device, analog, digital and mixed-signal. While the code currently only supports analog simulation, development is proceeding to support both the device-scale[2] and digital/mixed-signal modeling. This is done in a rigorous and tightly coupled manner, allowing for timely, full-system solutions.

Analysis capability

Xyce currently supports DC and transient as well as a variety of optimization and design options available from the DAKOTA optimization framework [3] . This document does not cover the coupling of **Xyce** with DAKOTA. Sandia customers may contact the **Xyce** team for assistance with this "developing" capability. Full support is expected in the next major release.

Object-oriented code design and implementation

Xyce was designed and written from the ground up utilizing modern coding practices to ensure the optimal combination of code performance, code maintenance and code extensibility. This design allows for rapid implementation of new capability as well as long term maintenance of the code.

2.3 Analysis Support within **Xyce**

Several simulation analysis options are supported within **Xyce**. For basic analysis, **Xyce** currently supports DC and transient analysis; AC analysis intended to be supported in a future release. Also, a variety of optimization and design options are available via coupling with the DAKOTA optimization framework [3]. While DAKOTA is not distributed as part of **Xyce**, Sandia customers may contact the **Xyce** team for assistance with this capability.

DC Analysis

DC analysis is used to evaluate the circuit response to a direct current input source. DC analysis is automatically performed prior to a transient analysis simulation in order to provide initial conditions. This is commonly referred to as the "DC Operating Point" calculation. In addition, DC analysis can be performed to provide additional information about the circuit. Table 2.1 summarizes the various DC analysis types and corresponding **Xyce** implementation.

[2] The work on the device-scale coupling has been demonstrated but is not supported in the current release. It is expected to be made generally available by the next major release.

DC Analysis Type...	Xyce Calculates...
DC sweep	Steady-state voltages and currents when sweeping a source, a model parameter, or temperature over a range of values.
Operating Point	Steady-state initial conditions for voltages and currents at a DC bias.

Table 2.1. DC Analysis Capabilities.

Transient Calculations

The transient analysis capability within **Xyce** computes the transient performance of a circuit for a specified time interval. The initial conditions are provided by a DC analysis (DC Operating Point) automatically performed at the beginning of the transient simulation.

Transient Analysis Type...	Xyce Calculates...
Transient	Voltages and current tracked over time.

Table 2.2. Transient Analysis Capabilities.

Optimization and Design

The DAKOTA framework [3] provides a suite of optimization and design tools that can be used in conjunction with **Xyce**. For more information, contact a member of the **Xyce** team (see the Contact information on page 5.)

3. Simulation Examples with **Xyce**

Chapter Overview

This chapter provides an introduction to the process and tools used to generate and run circuit design simulations and to evaluate the results. An example circuit is provided for each available analysis type.

3.1 Example Circuit Construction

This section describes how to use **Xyce** to create the simple diode clipper circuit shown in Figure 3.1.

While a schematic edit and capture capability is under development, **Xyce** currently only supports circuit creation via netlist editing. **Xyce** supports most of the standard netlist entries common to Berkeley SPICE 3F5 and Orcad PSpice. For users who are familiar with PSpice netlists, the differences between PSpice and **Xyce** netlists are listed in the **Xyce** Reference Guide [1].

Example: diode clipper circuit

1. Open a new netlist file using a standard text editor (e.g., VI, Emacs, notepad, etc.).
2. Type the title on the first line of the netlist:

```
Diode Clipper Circuit
```

3. Create a 5V DC voltage source between nodes 1 and 0 by typing the following on a new line:

```
VCC 1 0 5V
```

4. Create another DC voltage source between nodes 3 and 0 by entering the following on a new line:

```
VIN 3 0 0V
```

5. Place the diodes in the circuit between nodes 2 and 1, and nodes 0 and 2, respectively, by entering the following lines:

```
D1 2 1 D1N3940
D2 0 2 D1N3940
```

6. Enter resistors R1, R2, R3 and R4, respectively:

```
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
```

7. Place the capacitor in the circuit:

```
C1 2 4 0.47u
```

8. Add the diode model to the netlist to complete it as Figure 3.2.
9. Complete the netlist by entering `.END` on the last line in the file. Save the file as `clipper.cir`. The complete netlist is shown in Figure 3.2 and the schematic in Figure 3.1.

The netlist in Figure 3.2 illustrates some of the syntax of a netlist input file. Netlists begin with a title (e.g., "`Diode Clipper Circuit`"), support comments (lines beginning with the "`*`" character), devices, model definitions and the "`.END`" statement.

This netlist file is not yet complete and will not run properly using **Xyce** (see Section 3.2 for instructions on running **Xyce**) as it lacks an analysis statement. As you proceed in this chapter, you will see how to add the appropriate analysis statement and run the clipper circuit.

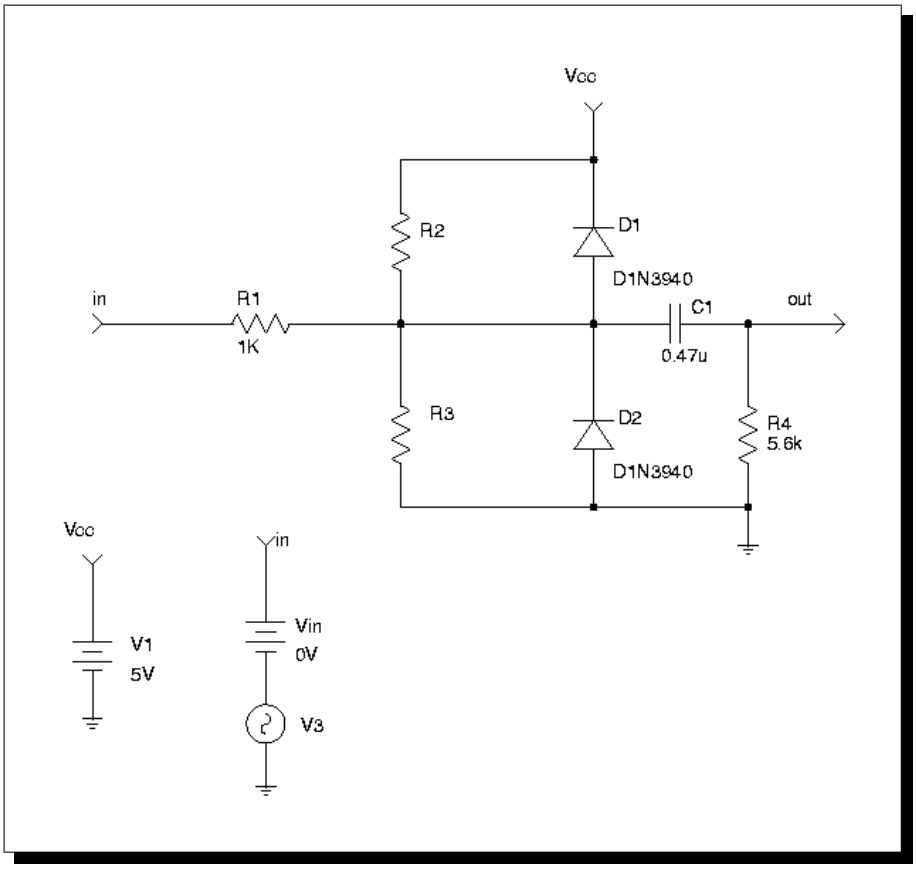

Figure 3.1. Schematic of diode clipper circuit with DC and transient voltage sources.

```
Diode Clipper Circuit
*
* Voltage Sources
VCC 1 0 5V
VIN 3 0 0V
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
*
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE
* SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
*
.END
```

Figure 3.2. Diode clipper circuit netlist.

3.2 Running **Xyce**

While a GUI for **Xyce** is under development, **Xyce** is currently run from the command line. This section provides an overview of how to run **Xyce**.

Command Line Operation

Running **Xyce** from the command line is straightforward. The scripts `xmpirun` and `runxyce` created during Installation (1.2) set up the runtime environment for you and execute **Xyce**. Depending on whether you are using a version compiled with MPI or a serial version, there are two simple ways to begin running **Xyce**:

- Running serial **Xyce**:

```
> runxyce [options] <netlist filename>
```

- Running **Xyce** in parallel:

```
> xmpirun -np <# procs> [options] <netlist filename>
```

While **Xyce** is running, the progress of the simulation is output to the command line window.

For a more complete description of running **Xyce** in serial and in parallel, see Section 6.3.

3.3 DC Sweep Analysis

This section illustrates how to run a DC sweep analysis using **Xyce**. In this example we examine the DC response of the clipper circuit by running a DC sweep of the input voltage source (`Vin`) and reviewing at the results generated by **Xyce**. This example demonstrates using DC sweep analysis parameters that vary `Vin` from -10 to 15 volts in 1 volt steps.

Example: DC sweep analysis

To set up and run a DC sweep analysis using the diode clipper circuit:

1. Open the diode clipper circuit netlist file (`clipper.cir`) using a standard text editor (e.g., VI, Emacs, notepad, etc.).
2. Enter the analysis control statement in the netlist:

```
.DC VIN -10 15 1
```

3. Enter the output control statement:

```
.PRINT DC V(3) V(2) V(4)
```

4. Save the netlist file and run **Xyce** on the circuit. For example, to run serial **Xyce**:

```
> runxyce clipper.cir
```

5. Open the results file (clipper.cir.prn) and examine (or plot) the output voltages that were selected for nodes 3 (Vin), 2 and 4 (Out). Figure 3.4 shows the output plotted as a function of the swept variable Vin.

The modified netlist is shown below in Figure 3.3.

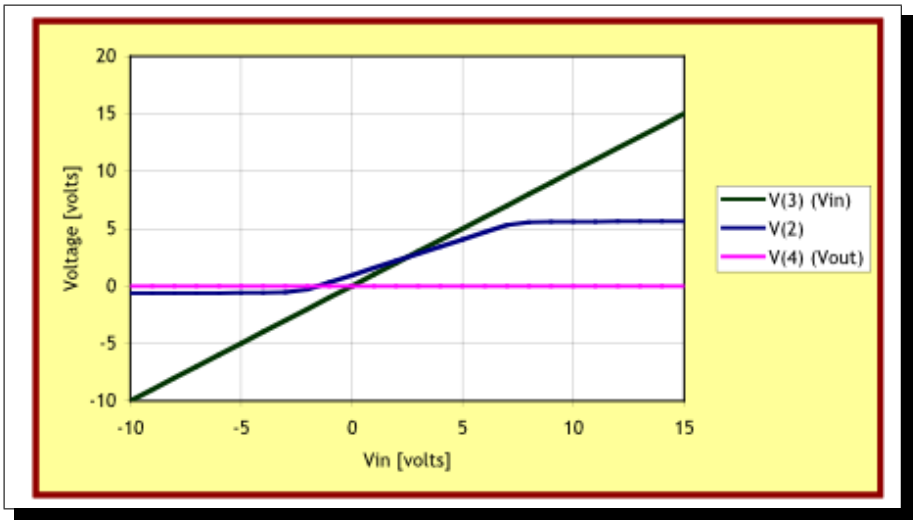

Figure 3.4. DC sweep voltages at Vin, node 2 and Vout.

Table 3.1 gives references for further explanation of the supported DC sweep analysis.

To find out more about...	See...
DC analysis for analog designs	Chapter 7, DC Analysis

Table 3.1. DC Analysis References

3.4 Transient Analysis

This section shows how to run a transient analysis using **Xyce**. In this example, we look at the transient response of the clipper circuit to a sinusoidal input voltage source (`Vin`)

```
Diode Clipper Circuit with DC sweep analysis statement
*
* Voltage Sources
VCC 1 0 5V
VIN 3 0 0V
* Analysis Command
.DC VIN -10 15 1
* Output
.PRINT DC V(3) V(2) V(4)
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
*
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE
* SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
*
.END
```

Figure 3.3. Diode clipper circuit netlist for DC sweep analysis.

and review the results generated by **Xyce**. This example utilizes a sinusoidal input voltage source running at a frequency of 1 kHz and amplitude of 10 volts. To set up this example, we must modify the netlist to include this source.

Example: transient analysis

To set up and run a transient analysis using the diode clipper circuit:

1. Open the diode clipper circuit netlist file file (clipper.cir) using a standard text editor (e.g., VI, Emacs, notepad, etc.).
2. If you added DC analysis statements in the previous example, remove them (see Figure 3.4).
3. Enter the analysis control in the netlist:

```
.TRAN 2ns 2ms
```

4. Modify the input voltage source (`Vin`) to generate the sinusoidal input signal:

```
VIN 3 0 SIN(10V 1kHz)
```

5. Save the netlist file and run **Xyce** on the circuit. For example, to run serial **Xyce**:

```
> runxyce clipper.cir
```

6. Open the results file and examine (or plot) the output voltages for nodes 3 (`Vin`), 2 and 4 (`Out`). The plot in Figure 3.6 shows the output plotted as a function of time.

The modified netlist is shown in Figure 3.5.

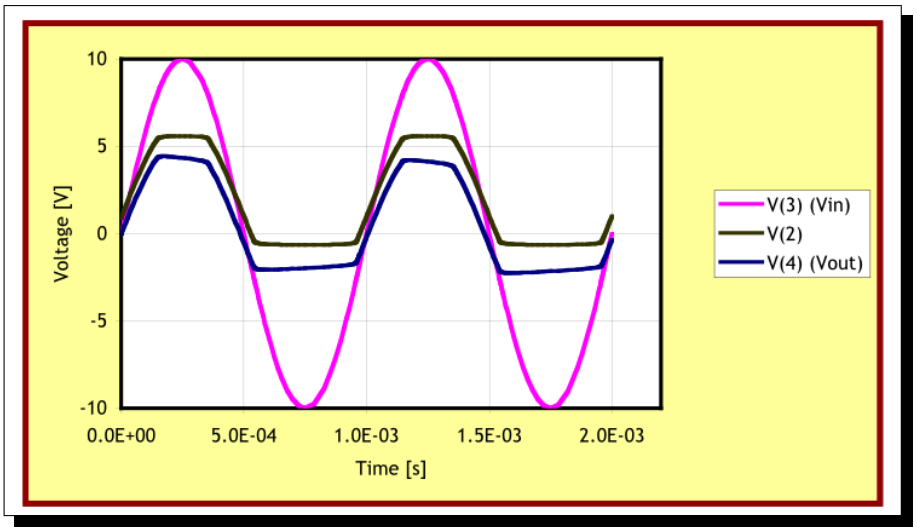

Figure 3.6. Sinusoidal input signal and clipped outputs.

```
Diode Clipper Circuit with transient analysis statement
*
* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
* Analysis Command
.TRAN 2ns 2ms
* Output
.PRINT TRAN V(3) V(2) V(4)
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
*
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE
* SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
*
.END
```

Figure 3.5. Diode clipper circuit netlist for transient analysis.

Table 3.2 below gives references for further explanation of the supported transient analysis.

To find out more about. . .	See. . .
Transient analysis for analog designs	Chapter 8, Transient Analysis

Table 3.2. Transient Analysis References.

4. Simulation Design Creation

Chapter Overview

This chapter contains basic information on creating circuit designs. Sections include:

- Section 4.1 *Netlist Circuit Description*
- Section 4.2 *Devices Available for Simulation*
- Section 4.3 *Parameters and Expressions*

4.1 Netlist Circuit Description

Netlist Overview

Using a netlist to describe a circuit for **Xyce** is the primary method used for running a circuit simulation. Netlist support within **Xyce** largely conforms to that used by Berkeley SPICE 3F5 with several new options for controlling functionality unique to **Xyce**. In a netlist, the circuit is described by a set of "element lines" which define the circuit elements and their values, the circuit topology (the connection of the circuit elements), and a variety of control options for the simulation. The first line in the netlist file must be a title and the last line must be "`.END`". Between these two constraints, the order of the statements is irrelevant.

Netlist Elements

An "element line", for which the format is determined by the specific element type, defines each circuit element instance. The general format is given by:

```
<type><name> <node information> <element information...>
```

The `<type>` must be a letter (A through Z) and the `<name>` follows immediately. For example, `RARESITOR` specifies a type=resistor with a name `ARESITOR`. Fields on a line are separated by spaces, commas, an equal sign or a left or right parenthesis.

A number field may be an integer or a floating-point value. Either one may be followed by one of the following scaling factors:

Symbol	Equivalent Value
T	10^{12}
G	10^{9}
Meg	10^{6}
K	10^{3}
mil	25.4^{-6}
m	10^{-3}
u (μ)	10^{-6}
n	10^{-9}
p	10^{-12}
f	10^{-15}

Node information is given in terms of node names, which are arbitrary character strings. The only requirement is that the ground node is named '0'. There are some restrictions on the circuit topology:

- There can be no loop of voltage sources and/or inductors.
- There can be no cut-set of current sources and/or capacitors.
- Every node must have a DC path to ground.
- Every node must have at least two connections (with the exception of unterminated transmission lines and MOSFET substrate nodes).

The following line provides an example of an element line that defines a resistor between nodes 1 and 3 with a resistance value of $10\ k\Omega$.

Example: `RARESISTOR 1 3 10K`

Title, Comments and End

The title line is required to be the first line in the input netlist and is included in the output file.

Example: `Test RLC Circuit`

The "`.End`" line must be the last line in the netlist.

Example: `.END`

Comments are supported in netlists and are indicated by placing an asterisk at the beginning of the comment line. They may occur anywhere in the netlist *but* they must be at the beginning of a line. **Xyce** also supports "in-line" comments. An in-line comment is designated by a semicolon and may occur on any line. Everything after the semicolon is taken as a comment and ignored. Any line that begins with leading white space is also considered to be a comment.

Example: `* This is a netlist comment.`

Example: `.DC .... * This type of inline comment is not supported.`

Example: `.DC .... ; This type of inline comment is supported.`

Netlist Commands

Command elements are used to describe the analysis being defined by the netlist. Examples include analysis types, initial conditions, device models and output control. The **Xyce** Reference Guide [1] contains a refernce for these commands.

Example: `.PRINT TRAN V(Vout)`

Analog Devices

The analog devices supported include most of the standard circuit components normally found in circuit simulators such as SPICE 3F5, PSpice, etc., plus several Sandia specific devices.

Example: `D_CR303 N_0065 0 D159700`

Table 4.1 below gives references for further explanation of the supported analog devices.

To find out more about...	See...
Analog devices	**Xyce** Reference Guide [1]

Table 4.1. Analog Devices References.

4.2 Devices Available for Simulation

This section describes the different types of analog devices supported in **Xyce**. These include standard analog devices, sources (dependent and independent) and subcircuits. Each device description has the following information:

- A description and an example of the netlist syntax
- The corresponding model types and descriptions, where applicable
- The corresponding lists of model parameters and descriptions, where applicable
- The associated circuit diagram and model equations (as necessary)

These analog devices include all of the standard circuit components needed for most analog circuits. User defined models may also be implemented using the `.MODEL` (model definition) statement and macromodels as subcircuits using the `.SUBCKT` (subcircuit) statement.

Analog Devices

Xyce supports many analog devices, including sources, subcircuits and behavioral models. The devices are classified into device types, each of which can have one or more model types. For example, the BJT device type has two model types: NPN and PNP.

The device element statements in the netlist always start with the name of the individual device instance. The first letter of the name determines the device type. The format of the following information depends on the device type and its parameters. The Device Type summary table (Table 4.2) lists all of the analog devices supported by **Xyce**. Each standard device is then described in more detail in the following sections. Except where noted, the devices are based upon those found in [4].

Table 4.2 is a summary of the analog device types and the form of their netlist formats. For a more complete description of the syntax for supported devices, see the **Xyce** Reference Guide. [1].

Device Type	Designator Letter	Typical Netlist Format
Capacitor	C	C<name> <+ node> <- node> [model name] <value> + [IC=<initial value>]
Inductor	L	L<name> <+ node> <- node> [model name] <value> + [IC=<initial value>]
Resistor	R	R<name> <+ node> <- node> [model name] <value> + [L=<length>] [W=<width>]
Diode	D	D<name> <anode node> <cathode node> + <model name> [area value]
Mutual Inductor	K	K<name> <inductor 1> [<ind. n>*] + <linear coupling or model>
Independent Voltage Source	V	V<name> <+ node> <- node> [[DC] <value>] + [AC <magnitude value> [phase value]] + transient specification]
Independent Current Source	I	I<name> <+ node> <- node> [[DC] <value>] + [AC <magnitude value> [phase value]] + [transient specification]
Voltage Controlled Voltage Source	E	E<name> <+ node> <- node> <+ controlling node> + <- controlling node> <gain>
Voltage Controlled Current Source	G	G<name> <+ node> <- node> <+ controlling node> + <- controlling node> <transconductance>
Nonlinear Dependent Source (B Source)	B	B<name> <+ node> <- node> + <I or V>={<expression>}
Bipolar Junction Transistor (BJT)	Q	Q<name> <collector node> <base node> <emitter node> [substrate node] <model name> [area value]

Device Type	Designator Letter	Typical Netlist Format
MOSFET	M	M<name> <drain node> <gate node> <source node> + <bulk/substrate node> <model name> + [common model parameter]*
Transmission Line	T	T<name> <A port + node> <A port - node> + <B port + node> <B port - node> + <ideal specification>
Voltage Controlled Switch	S	S<name> <+ switch node> <- switch node> + <+ controlling node> <- controlling node> + <model name>
Subcircuit	X	X<name> [node]* <subcircuit name> + [PARAMS:[<name>=<value>]*]
PDE Devices	Z	Z<name> <node1> <node2> [node3] + [node4] <model name>

Table 4.2: Analog Device Quick Reference.

4.3 Parameters and Expressions

In addition to explicit values, the user may use parameters and expressions to symbolize numeric values in the circuit design.

Parameters

A parameter is like a programming variable that represents a numeric value by name. Once you have defined a parameter (declared its name and given it a value) at a particular level in the circuit hierarchy, you can use it to represent circuit values at that level or any level directly beneath it in the circuit hierarchy. One way that you can use parameters is to apply the same value to multiple part instances.

How to Declare and Use Parameters

In order to use a global parameter in a circuit, one must:

- define the parameter using a `.PARAM` statement within a netlist
- replace an explicit value with the parameter in the circuit

Note that **Xyce** reserves several keywords that may not be used as parameter names. These are:

- Time
- Vt
- Temp
- GMIN

However, in this first release of **Xyce**, only Time is predefined.

Example: Declaring a parameter

1. Locate the level in the circuit hierarchy at which the .PARAM statement declaring a parameter will be placed (note: a global parameter that can be used anywhere in the netlist can be declared by placing the .PARAM statement at the top-most level of the circuit).
2. Name the parameter and give it a value. The value can be numeric or given by an expression:

```
.SUBCKT subckt1 n1 n2 n3
.PARAM res = 100
*
* other netlist statements here
*
.ENDS
```

3. Note: the parameter "res" can be used anywhere within the subcircuit subckt1 including subcircuits defined within it, but cannot be used outside of subckt1.

Example: Using a parameter in the circuit

1. Find the numeric value that is to be replaced by a parameter: a device instance parameter value, model parameter value, etc. The value being replaced must be accessible with the current hierarchy level.
2. Replace the numeric value with the parameter name contained within braces ({}) as in:

```
R1 1 2 {res}
```

Expressions

In **Xyce**, an expression is a mathematical relationship that may be used any place one would use a number (numeric or boolean). **Xyce** evaluates the expression to a value when it reads in a new circuit netlist.

To use an expression in a circuit netlist:

1. Locate the value to be replaced (component, model parameter, etc.).
2. Substitute the value with an expression utilizing the $\{\}$ syntax:

   $\{$*expression*$\}$

   where *expression* can contain any of the following:

   - available operators from those in Table 4.3
   - included functions from those in Table 4.4
   - user-defined functions
   - the system variable `TIME` for use only in ABM expressions, see Chapter5.3
   - user-defined parameters that are within scope
   - literal operands

   The braces ($\{\}$) instruct **Xyce** to evaluate the expression and use the resulting value.

Example: Using an expression

Scaling the DC voltage of a $12V$ independent voltage source, designated `VF`, by some factor can be accomlished by the following netlist statements (in this example the factor is 1.5):

```
.PARAM FACTORV=1.5
VF 3 4 {FACTORV*12}
```

Xyce will evaluate the expression to:

`12 * 1.5` or `18` volts

[1] Logical and relational operators are used only with the `IF()` function.

Class of operator...	Operator...	Meaning
arithmetic	+	addition or string concatenation
	-	subtraction
	*	multiplication
	/	division
	**	exponentiation
logical[1]	~	unary NOT
	|	boolean OR
	^	boolean XOR
	&	boolean AND
relational	==	equality
	!=	non-equality
	>	greater-than
	>=	greater-than or equal
	<	less-than
	<=	less-than or equal

Table 4.3. Expression operators

Function...	Meaning...	Explanation...
ABS(x)	$\|x\|$	
SQRT(x)	$\sqrt{x}$	
MIN(x,y)	$\min(x, y)$	minimum of x and y
MAX(x,y)	$\max(x, y)$	maximum of x and y
EXP(x)	e^x	
LN(x)	$\ln(x)$	log base e
LOG(x)	$\log(x)$	log base 10
SIN(x)	$\sin(x)$	x in radians
ASIN(x)	$\arcsin(x)$	result in radians
SINH(x)	$\sinh(x)$	x in radians
ASINH(x)	$\sinh^{-1}(x)$	result in radians
COS(x)	$\cos(x)$	x in radians
ACOS(x)	$\arccos(x)$	result in radians
COSH(x)	$\cosh(x)$	x in radians
ACOSH(x)	$\cosh^{-1}(x)$	result in radians
TAN(x)	$\tan(x)$	x in radians
ATAN(x)	$\arctan(x)$	result in radians
TANH(x)	$\tanh(x)$	x in radians
ATANH(x)	$\tanh^{-1}(x)$	result in radians
ATAN2(x,y)	$\arctan(y/x)$	result in radians
SGN(x)	+1 if $x > 0$ 0 if $x = 0$ -1 if $x < 0$	
STP(x)	1 if $x > 0$ 0 otherwise	suppress a value until a given time
URAMP(x)	x if $x > 0$ 0 otherwise	
IF(t,x,y)	x if t is true, y otherwise	t is an expression using the relational operators in Table 4.3
DDT(x)	time derivative of x	
SDT(x)	time integral of x	

Table 4.4. Functions in arithmetic expressions

5. Working with Models

Chapter Overview

This chapter contains model ideas and a summary of the ways to create and modify models. Sections include:

- Section 5.1, *Definition of a Model*
- Section 5.2, *Model Organization*
- Section 5.3, *Analog Behavioral Modeling*

5.1 Definition of a Model

A model describes the electrical performance of a *part*. A part is a component in the circuit with *specific simulation properties* that *define* the part. In a netlist, a part is identified by its implementation properties designated by the associated model name.

Depending on the given device type, a model is defined as either:

- a model parameter set
- a subcircuit netlist

Both methods of defining a model use a netlist format, with precise syntax rules as described below.

Defining models using model parameters

Xyce currently has no built-in models. However, models can be defined for a device by changing some or all of the *model parameters* from their defaults via the `.MODEL` syntax. For example:

Example: `.MODEL MLOAD1 NMOS (LEVEL=2 VTO=0.5 CJ=0.025pF)`

Defining models using subcircuit netlists

In **Xyce**, models may also be defined using the *subcircuit syntax:* `.SUBCKT`/`.ENDS`. This syntax includes:

- *netlists* to define the configuration and function of the part.
- *variable input parameters* to modify the model.

See Figure 5.1 for an example.

```
*** SUBCIRCUIT: l3dsc1
*** Parasitic Model: microstrip
*** Only one segment
.SUBCKT l3dsc1 1 3 2 4
C01 1 0 4.540e-12
RG01 1 0 7.816e+03
L1 1 5 3.718e-08
R1 5 2 4.300e-01
C1 2 0 4.540e-12
RG1 2 0 7.816e+03
C02 3 0 4.540e-12
RG02 3 0 7.816e+03
L2 3 6 3.668e-08
R2 6 4 4.184e-01
C2 4 0 4.540e-12
RG2 4 0 7.816e+03
CM012 1 3 5.288e-13
KM12 L1 L2 2.229e-01
CM12 2 4 5.288e-13
.ENDS
```

Figure 5.1. Example subcircuit model.

Subcircuit Hierarchy

Xyce supports the definition of subcircuits within other subcircuits. Each subcircuit definition introduces a new level in the circuit hierarchy with the top level begin the main circuit. If a second level is defined, it is composed of the subcircuits in the main circuit and each subsequent level is composed of the subcircuits contained in the previous level. A subcircuit may also contain other definitions such as models via the `.MODEL` statement, parameters via the `.PARAM` statement, and functions via the `.FUNC` statement.

In this context, the subcircuit defines the "scope" for the definitions it contains. That is, *the definitions contained within a subcircuit can be used within that subcircuit and/or within any subcircuit it contains*. Any definitions occuring in the main circuit have global scope and can be used anywhere in the circuit. A name, such as a model, parameter, function or subcircuit name, occurring in a definition at one level of a circuit hierarchy can be redefined at any lower level contained directly by the subcircuit. In this case, the new definition applies at the given level and those below.

In the following example, the model named `MOD1` can be used in subcircuits `SUB1` and `SUB2`

but not in the subcircuit SUB3. The parameter P1 has a value of 10 in subcircuit SUB1 and a value of 20 in subcircuit SUB2.

```
.SUBCKT SUB1 1 2 3 4
.MODEL MOD1 NMOS(LEVEL=2)
.PARAM P1=10
*
* subcircuit devices omitted for brevity
*
.SUBCKT SUB2 1 3 2 4
.PARAM P1=20
*
* subcircuit devices omitted for brevity
*
.ENDS
.ENDS

.SUBCKT SUB3 1 2 3 4
*
* subcircuit devices omitted for brevity
*
.ENDS
```

Figure 5.2. Example subcircuit model.

5.2 Model Organization

The organization of models entails the following fundamental concepts:

- model definitions are saved in model library files.
- model libraries must be constructed so that **Xyce** will look to them for model definitions.

Model libraries

Device model and subcircuit definitions are organized into model libraries. These libraries are text files (similar to netlist files) that have one or more model definitions. Model library names usually end with a `.lib` extension.

As a rule-of-thumb, model libraries files typically include similar model types. In these files, the *header comments* describe the models therein.

Model library configuration

Xyce searches the model library files for the model names given by the `.MODEL` statement for parts in the netlist. It then uses these model definitions in the simulation. For **Xyce** to be able to find these model definitions, the libraries must be "included" in the netlist. This is accomplished by adding a `.INCLUDE` statement to the netlist. As an example:

```
*** SUBCIRCUIT: l3dsc1
*** Parasitic Model: microstrip
*** Only one segment
.SUBCKT l3dsc1 1 3 2 4
.INCLUDE "model.lib"
C01 1 0 4.540e-12
RG01 1 0 7.816e+03
L1 1 5 3.718e-08
R1 5 2 4.300e-01
C1 2 0 4.540e-12
RG1 2 0 7.816e+03
C02 3 0 4.540e-12
RG02 3 0 7.816e+03
L2 3 6 3.668e-08
R2 6 4 4.184e-01
C2 4 0 4.540e-12
RG2 4 0 7.816e+03
CM012 1 3 5.288e-13
KM12 L1 L2 2.229e-01
CM12 2 4 5.288e-13
.ENDS
```

The scoping rules for `.INCLUDE` statements is the same as for other types of definitions as outlined in the preceding section.

5.3 Analog Behavioral Modeling

Overview of Analog Behavioral Modeling

The analog behavioral modeling capability of **Xyce** provides for flexible descriptions of electronic components in terms of a transfer function or lookup table. In other words,

a mathematical relationship is used to model a circuit segment removing the need for component by component design.

In **Xyce**, the `B` nonlinear dependent source device type is used for analog behavioral modeling . The PSpice equivalent is its `E` and `G` voltage controlled source device types. **Xyce** automatically converts some forms of the PSpice `E` and `G` devices to the equivalent `B` device as described in "Quick Reference for PSpice Users" section of the **Xyce** Reference Guide [1]. The manual conversion from a PSpice analog behavioral model is also be described in the **Xyce** Reference Guide.

Specifying ABM Devices

ABM devices (`B` devices) are specified in a netlist the same way as other devices. Customizing the operational behavior of the device is achieved by defining an ABM expression describing how inputs are transformed into outputs.

Device and node names in ABM expressions

ABM expressions follow the same rules as other expressions in a netlist with the additional ability to specify signals (node voltages and voltage source currents) in the expression. In ABM expressions, refer to signals by name. **Xyce** recognizes the following constructs in ABM expressions:

- `V(<node name>)`
- `V(<node name>,<node name>)`
- `I(<voltage source name>)`

In a hierarchical circuit (a circuit with possibly nested levels of subcircuits), voltage source names that appear in an ABM expression must be the name of a voltage source in the same subcircuit as the ABM device. Similarly, node names in an ABM expression must be the node names of one or more devices in the same subcircuit as the ABM device.

6. Creating and Running Analysis

Chapter Overview

This chapter outlines methods for creating and running different types of circuit analysis using **Xyce**. This is given in the following sections:

- Section 6.1, *Types of Analysis*
- Section 6.2, *Analysis Creation*
- Section 6.3, *Running a* **Xyce** *Simulation*
- Section 12.3, *Parallel Partitioning Options*

6.1 Types of Analysis

Currently **Xyce** supports only DC and transient analysis with plans to support AC analysis and a variety of design analysis types (e.g. design optimization, sensitivity analysis) in the future.

DC Analysis

DC analysis is used to evaluate the circuit response to a direct current input source. DC analysis is automatically performed prior to a transient analysis simulation in order to provide initial conditions. In addition, DC analysis can be performed in a stand-alone manner to provide additional information about the circuit. Currently, only the DC sweep analysis is supported which calculates steady-state voltages and currents when sweeping a source or a model parameter.

Transient Calculations

The transient analysis capability within **Xyce** computes the transient performance of a circuit for a specified time interval. The initial conditions are provided by a DC analysis automatically performed at the beginning of the transient simulation. From this, **Xyce** computes the transient response from `TIME=0` to a specified time. During this calculation, the **Xyce** time integrating algorithms adjust the calculation's time step size to correspond to the circuit's response while minimizing the overall number of time steps. This is accomplished by ensuring the default or user specified error requirements while maximizing the time-step size at any point. During fast changing portions of the simulation, the time-step size is reduced, while during less active portions, the step-size is increased.

6.2 Analysis Creation

To set up a simulation analysis:

1. Open a new or existing netlist text file using a text editor (e.g., XEmacs, VI, Notepad). See Figure 6.1 for an example of editing a netlist using XEmacs.
2. Enter the netlist commands that describe the circuit, analysis type and output information. See Chapter 3 for a description on how to setup a circuit design.
3. Enter the required parameter values and other information to complete the analysis specifications.
4. Save the file for running in **Xyce**.

Specific information for setting up each type of analysis is discussed in the following chapters.

6.3 Running a **Xyce** Simulation

While a GUI for **Xyce** is under development and will be part of an overall simulation framework, **Xyce** is currently run from the command line. This section outlines how to run **Xyce** for both serial and MPI parallel simulations.

Command Line Simulation

Running **Xyce** from the command line is straightforward. The scripts `xmpirun` and `runxyce` created during installation (see Section 1.2) set up the runtime environment and execute **Xyce**. (Microsoft Windows users should use the `runxyce.bat` batch file.) Depending on whether you are using a version compiled with MPI support or a serial version, there are two ways to begin running **Xyce**:

- Running serial **Xyce**:

```
> runxyce [options] <netlist filename>
```

- Running **Xyce** in parallel:

```
> xmpirun -np <# procs> [options] <netlist filename>
```

where [`options`] are the command line arguments for **Xyce**. For example, to log output to a file named `sample.log` type:

```
$ runxyce -l sample.log <netlist filename>
```

The next example runs parallel **Xyce** on four processors and places the results into a comma separated value file named `results.csv`:

```
$ xmpirun -np 4 -delim COMMA -o results.csv <netlist filename>
```

These examples assume that `<netlist filename>` is either in the current working directory or includes the path (full or relative) to the netlist file. Enclose the filename in quotation marks (" ") if the path contains spaces. Help is accessible with the `-h` option.

For MPI runs, [`options`] may also include command line arguments to `mpirun`. Consult the documentation installed with MPI on your platform for more details on MPI options. The `-np <# procs>` denotes the number of processors to use for the simulation. *NOTE: It is critical that the number of processors used is less than the number of devices and voltage nodes in the netlist.* The appropriate script used to run **Xyce** for each supported platform is listed in the Table 6.2.

Computer Architecture	OS	Serial Executable	MPI Executable
Apple PPC	OSX	runxyce	Not Available
Compaq	DEC OSF		xmpirun
SGI 64 bit	IRIX 6.5		
Intel X86	Linux		
Intel X86	FreeBSD		
Intel X86	Microsoft Windows 2000	runxyce.bat	Not Available

Figure 6.2. Platform scripts for running **Xyce**.

While **Xyce** is running, the progress of the simulation is output to the command line window. See the **Xyce** Reference Guide for complete list and explanation of command line options.

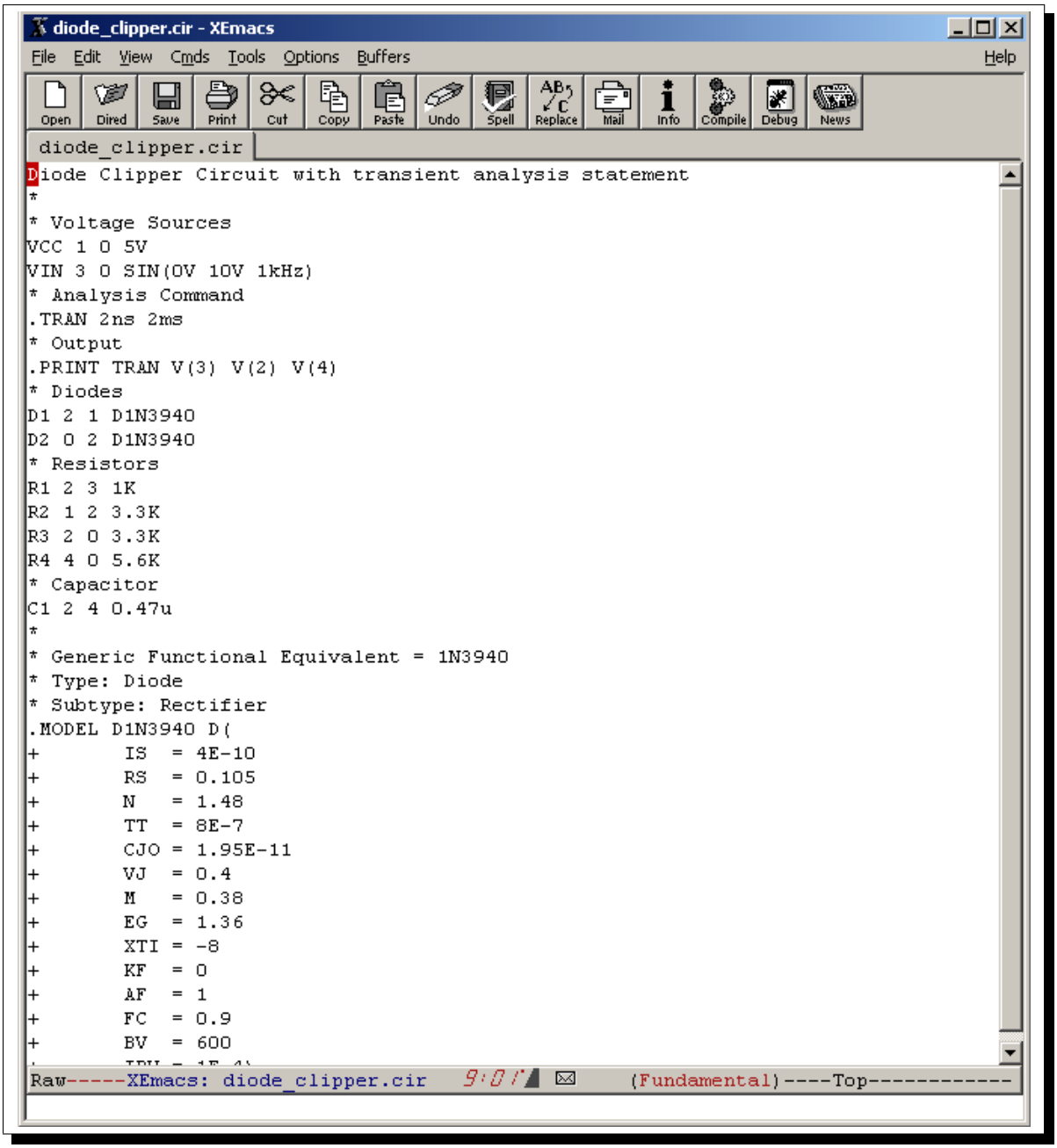

Figure 6.1. Example netlist editing using XEmacs.

7. DC Analysis

Chapter Overview

This chapter explains how to set up DC analysis and includes the following sections:

- Section 7.1, *Overview of DC Sweep*
- Section 7.2, *Setting Up and Running a DC Sweep*

7.1 Overview of DC Sweep

The DC sweep analysis capability in **Xyce** carries out a sweep, in DC mode, on a circuit. DC sweep is supported for a source (current or voltage), through a range of specified values. As the sweep proceeds, the bias point is computed for each value in the specified range of the sweep.

If the variable to be swept is a voltage or current source, a DC source must be used. The DC value is set in the netlist (see the **Xyce** Reference Guide [1]. In simulating the DC response of an analog circuit, **Xyce** eliminates any time dependence from the circuit. This is accomplished by treating all capacitor elements as open circuits, all inductor elements as short circuits and using only the DC values of both voltage and current sources.

7.2 Setting Up and Running a DC Sweep

Following the example given in Section 3.3, the diode clipper circuit netlist is shown in Figure 7.1 with a DC sweep analysis specified. Here, the voltage source `Vin` is swept from -10 to 15 in 1 volt increments, resulting in 26 DC operating point calculations. Note also that the default setting for `Vin` is ignored during these calculations. All other source values use the specified values (`VCC = 5V` in this case).

Running **Xyce** on this netlist produces an output results file named `clipper.cir.prn`. Plotting this data produces the graph shown in Figure 7.2.

7.3 OP Analysis

Xyce also supports `.OP` analysis statements. In **Xyce**, `.OP` should be considered as a shorthand for a single step DC sweep, in which all the default operating point values are used. One can also consider `.OP` analysis to be the operating point calculation which would occur as the intitial step to a transient calculation, without the subsequent time steps.

This capability was mainly added so that the code would be able to handle legacy netlists which used this type of analysis statement. In most versions of SPICE, using `.OP` will result in extra output which is not available from a DC sweep. That additional output capability has not yet been implemented in **Xyce**.

```
Diode Clipper Circuit
** Voltage Sources
VCC 1 0 5V VIN 3 0 0V
* Analysis Command
.DC VIN -10 15 1
* Output
.PRINT DC V(3) V(2) V(4)
* Diodes
D1 2 1 D1N3940 D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
** GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE
* SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
* .END
```

Figure 7.1. Diode clipper circuit netlist for DC sweep analysis.

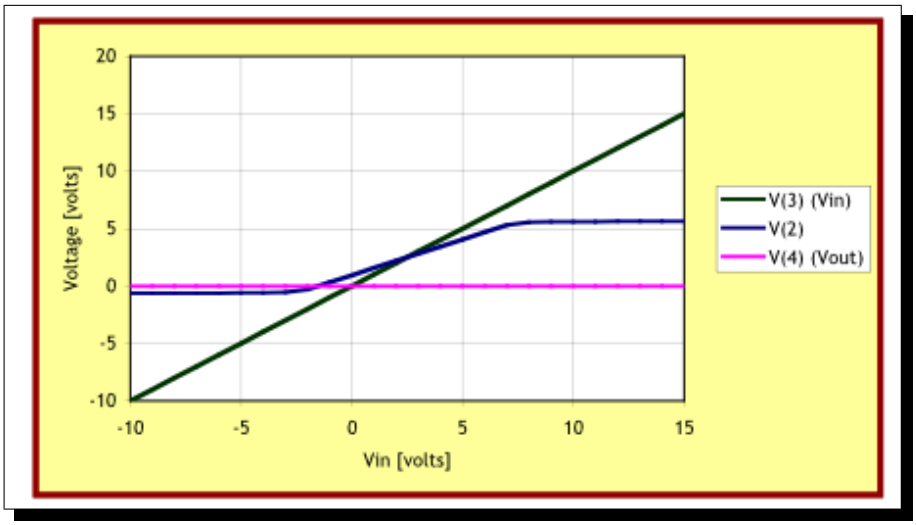

Figure 7.2. DC sweep voltages at Vin, node 2 and Vout.

8. Transient Analysis

Chapter Overview

This chapter illustrates how to set up a transient analysis and includes the following sections:

- Section 8.1, *Transient Analysis Overview*
- Section 8.2, *Defining a Time-Dependent Source*
- Section 8.3, *Transient Calculation Time Steps*
- Section 8.4, *Checkpointing and Restarting*

8.1 Transient Analysis Overview

The transient response analysis simulates the response of the circuit from `TIME=0` to a specified time. Throughout a transient analysis, any or all of the independent sources may have time-dependent values.

In **Xyce**, the transient analysis begins by performing its own bias point calculation at the beginning of the run, using the same method as used for DC sweep. This is required to set the initial conditions for the transient solution as the initial values of the sources may differ from the their DC values.

To run a transient simulation, the circuit netlist file must include a `.TRAN` command with the parameters required for the desired transient analysis (see the **Xyce** Reference Guide [1]). In addition, the netlist must contain one of the following:

- an independent, transient source (see Table 8.1),
- an initial condition on a reactive element, or
- a time-dependent behavioral modeling source (see Chapter 5.3)

8.2 Defining a Time-Dependent (transient) Source

Overview of Source Elements

Source elements, either voltage or current, are entered in the netlist file as described in the **Xyce** Reference Guide [1]. Table 8.1 list the time-dependent sources available in **Xyce** for either voltage or current. For voltage sources, the name is preceeded by the letter `V` while current sources are preceeded by the letter `I`.

To use one of these time-dependent or transient sources, the user must place the source element line in the netlist and characterize the transient behavior using the appropriate parameters. Each transient source element has a separate set of parameters dependent on its transient behavior. In this way, the user can create analog sources which produce sine wave, square pulse, exponential pulse, single-frequency FM, and piecewise linear waveforms.

Defining Transient Sources

To define a transient source:

Source Element Name	Description
EXP	Exponential Waveform
PULSE	Pulse Waveform
PWL	Piecewise Linear Waveform
SFFM	Frequency-modulated Waveform
SIN	Sinusoidal Waveform

Table 8.1. Summary of time-dependent sources supported by **Xyce**.

- Select one of the supported sources: independent voltage or current source.
- Choose a transient source type from Table 8.1.
- Provide the transient parameters (see the **Xyce** Reference Guide [1]) to fully define the source.

Below is an example of an independent sinusoidal voltage source in a circuit netlist. It creates a voltage source between nodes 1 and 5 that oscillates sinusoidally between -5V and +5V with a frequency of 50 KHz.

Example: `Vexample 1 5 SIN(-5V 5V 50KHz)`

8.3 Transient Calculation Time Steps

During the simulation, **Xyce** uses a calculation time step that is continuously adjusted for accuracy and efficiency (see [5]). During periods of circuit idleness the calculation time step is increased, and during dynamic portions of the waveform it is decreased. This release of **Xyce** does not allow the user to specify a maximum time step.

The internal calculation time steps used might not be consistent with the output time steps requested by the user. By default **Xyce** outputs solution results at every time step it calculates. If the user selects output timesteps via the `.OUTPUT` statement (see Chapter 11) then **Xyce** will output results for the closest time step that follows the time requested by the user. There is currently no mechanism for forcing **Xyce** to output at precise user-specified times.

8.4 Checkpointing and Restarting

The .OPTIONS RESTART command (in the netlist) is used to control all checkpoint output and restarting. Checkpointing and associated restart can be extremely useful for long simulations. In essence, **Xyce** allows the user to save the state of the simulation during a run (at intervals the user specifies) (*checkpointing*). This checkpoint data can then be read in to *restart* the simulation from any of the saved (*checkpointed*) time points.

Checkpointing Command Format

- .OPTIONS RESTART PACK=<0|1> JOB=<job name> [INITIAL_INTERVAL=<interval> [<t0> <i0> [<t1> <i1>...]]]

  PACK=<0|1> indicates whether the restart data files will contain byte packed data or not. JOB=<job name> identifies the prefix for restart files. The actual restart files will be the job name appended with the current simulation time (e.g. name1e-05 for JOB=name and simulation time 1e-05 seconds). Furthermore, the INITIAL_INTERVAL=<interval> identifies the initial interval time used for restart output. The <tx ix> intervals identify times (tx) at which the output interval (ix) will change. This functionality is identical that described for the .OPTIONS OUTPUT command (see Section 11.1).

- Example - generate checkpoints at every time step (default):

```
.OPTIONS RESTART JOB=checkpt
```

- Example - generate checkpoints every 0.1 μs:

```
.OPTIONS RESTART JOB=checkpt INITIAL_INTERVAL=0.1us
```

- Example - generate unpacked checkpoints every 0.1 μs:

```
.OPTIONS RESTART PACK=0 JOB=checkpt INITIAL_INTERVAL=0.1us
```

- Example - Initial interval of 0.1 μs, at 1 μs in the simulation, change to interval of 0.5 μs, and at 10 μs change to an interval of 0.1 μs:

```
.OPTIONS RESTART JOB=checkpt INITIAL_INTERVAL=0.1us 1us 0.5us
+ 10us 0.1us
```

Restarting Command Format

- .OPTIONS RESTART <FILE=<filename> | JOB=<job name> START_TIME=<time>>
  + [INITIAL_INTERVAL=<interval> [<t0> <i0> [<t1> <i1> ...]]]

To restart from an existing restart file, the file can be specified by using either the FILE=<filename> parameter to explicitly request a file or JOB=<job name> START_TIME=<time> to specify a file prefix and a specific time. The time must exactly match an output file time for the simulator to correctly load the file. To continue generating restart output files, INITIAL_INTERVAL=<interval> and following intervals can be appended to the command in the same format as described above.

- Example - Restart from checkpoint file at 0.133 μs:

```
.OPTIONS RESTART JOB=checkpt START_TIME=0.133us
```

- Example - Restart from checkpoint file at 0.133 μs :

```
.OPTIONS RESTART FILE=checkpt0.000000133
```

- Example - Restart from 0.133 μs and continue checkpointing at 0.1 μs intervals:

```
.OPTIONS RESTART FILE=checkpt0.000000133 JOB=checkpt_again
+ INITIAL_INTERVAL=0.1us
```

9. STEP Parametric Analysis

Chapter Overview

This chapter illustrates how to set up a step analysis and includes the following sections:

- Section 9.1, *STEP Parametric Analysis Overview*
- Section 9.2, *Stepping over Instance Parameters*
- Section 9.3, *Stepping over Model Parameters*
- Section 9.4, *Stepping over Temperature*
- Section 9.5, *Special Cases: Independent Sources, etc.*

9.1 STEP Parametric Analysis Overview

The .STEP command performs a parametric sweep for all the analysies of the circuit. When this command is invoked, all of the typical analysis, such as .DC or .TRAN analysis are performed at each parameter step.

This capability is very similar to the STEP capability in PSPICE and ChileSPICE, but not identical. Efforts will be made in future releases to make the .STEP capability in Xyce 100% compatible with those codes. In **Xyce**, .STEP can be used to sweep over any device instance or device model parameter, as well as the circuit temperature. Currently, there is not a capability for sweeping global parameters, as specified by a .PARAM statement.

9.2 Sweeping over a Device Instance Parameter

One specifies a .STEP analysis by simply adding a .STEP line to a netlist. .STEP by itself is not an adequate analysis specification, as it merely specifies an outer loop around the normal analysis. There needs to be a standard analysis line, such as .TRAN or .DC as well.

A typical .STEP line looks like this:

Example:

```
.STEP M1:L 7u 5u -1u
```

This has a very similar format to the .DC line. In this example, M1:L is the name of the parameter, 7u is the initial value of the parameter, 5u is the final value of the parameter, and -1u is the step size. Currently, **Xyce** can only handle linear sweeps.

The example uses M1:L as the parameter, but it could have been any model or instance parameter that existed in the circuit. Internally, **Xyce** handles the parameters for all device models and device instances in the same way. You can uniquely identify any parameter by specifying the device instance name, followed by a colon (:), followed by the specific parameter name. For example, all the MOSFET models have an instance parameter for the channel length, L. If you have a MOSFET instance specified in a netlist, named M1, then the full name for M1's channel length parameter is M1:L.

A simple application of .STEP to a device instance is given in figure 9.1. This is the same diode clipper circuit as was used in the transient analysis chapter, except that a single line (in red font) has been added. The .STEP line will cause **Xyce** to sweep the resistance of the resistor, R4, from 3.0 KOhms to 15.0 KOhms, in increments of 2.0 KOhms. This means

that a total of seven transient simulations will be performed, each one with a different value for R4.

As the circuit is executed multiple times, the file output needs to be a little more sophisticated. The .PRINT statement is still used in much the same way as before. However, there is a separate *.prn output file for each .STEP increment.

The naming convention for .STEP simulation *.prn files is the same as in the non-.STEP case, except that the string "STEP*" is added to the name, where the "*" is an integer number indicating the step.

The example file given in figure 9.1 has a filename of clip.cir. The output files generated by the .PRINT statement are:

```
clip.cir.STEP0.prn  for R4    = 3.0K
clip.cir.STEP1.prn  for R4    = 5.0K
clip.cir.STEP2.prn  for R4    = 7.0K
clip.cir.STEP3.prn  for R4    = 9.0K
clip.cir.STEP4.prn  for R4   = 11.0K
clip.cir.STEP5.prn  for R4   = 13.0K
clip.cir.STEP6.prn  for R4   = 15.0K
```

These files will be similar in length, but not identical, as changing the resistance changes the numerical requirements a little bit, resulting in slightly different time step sizes.

9.3 Sweeping over a Device Model Parameter

Sweeping a model parameter can be done in an indentical manner to an instance parameter. Figure 9.2 contains the same circuit as in figure 9.1, but with a new .STEP line added. The new .STEP line refers to a model parameter, D1N3940:IS. Note that separate .STEP lines is the correct way to specify multiple parameters for .STEP analysis. Each parameter needs its own separate line. In this respect, the .STEP line syntax differs from the .DC line syntax.

9.4 Sweeping over Temperature

It is also possible to sweep over temperature. To do so, simply specify temp as the parameter name. It will work in the same manner as .STEP when applied to model and instance parameters.

```
Transient Diode Clipper Circuit with step analysis
* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
* Analysis Command
.TRAN 2ns 2ms
* Output
.PRINT TRAN V(3) V(2) V(4)
 * Step statement
.STEP R4:R 3.0K 15.0K 2.0K
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
*
.END
```

Figure 9.1. Diode clipper circuit netlist for step transient analysis.

```
Transient Diode Clipper Circuit with step analysis
* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
* Analysis Command
.TRAN 2ns 2ms
* Output
.PRINT TRAN V(3) V(2) V(4)
 * Step statements
.STEP R4:R 3.0K 15.0K 2.0K
.STEP D1N3940:IS 2.0e-10 6.0e-10 2.0e-10
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
*
.END
```

Figure 9.2. Diode clipper circuit netlist for 2-step transient analysis.

9.5 Special cases: Sweeping Independent Sources, Resistors, Capacitors

For some devices, there is generally only one parameter that one would want to actually sweep. For example, a linear resistor's only parameter of interest is the resistance, R. Similarly, for a DC voltage or current source, one is usually only interested in the magnitude of the source. Finally, linear capacitors generally only have the capacitance, C, as a parameter of interest. To make things easier for the user, these three types of devices have default parameters. Examples of usage are given below.

Example:

```
.STEP R4   3.0K   15.0K   2.0K
.STEP VCC  4.0     6.0    1.0
.STEP ICC  4.0     6.0    1.0
.STEP C1   0.45u   0.50u  0.1u
```

Independent sources require some extra explanation. There are number of different types of independent source, and only some of them have default parameters. Sources which are subject to `.DC` sweeps (swept sources) do not have a default parameter, as this could easily lead to infinite loops. The various independent source defaults are defined in the table.

Source Type	Default
Sinusoidal source	V0 (DC value, Offset)
Exponential source	V1 (DC value, Initial value)
Pulsed source	V2 (Pulsed value)
Constant, or DC source	V0 (Constant value)
Piecewise Linear source	No default
SFFM source	No default
Swept source (specified on a `.DC` line)	No default

Table 9.1: Default parameters for independent sources.

10. Using Homotopy Algorithms to Obtain Operating Points

Chapter Overview

This chapter includes the following sections:

10.1 Homotopy Algorithms Overview

The most difficult type of numerical nonlinear circuit problem to solve is a DC operating point. Unlike transient analysis, DC operating point analysis cannot rely on the results of a previous time step. Also, operating points often have multiple solutions, both valid and invalid.

Homotopy methods can often provide solutions to difficult nonlinear problems when other, more conventional numerical methods fail. In recent years, these techniques have been applied to circuit analysis. As of the **Xyce** Version 2.0 release, some of these algorithms have been added to **Xyce**. This chapter gives an introduction to the usage of homotopy algorithms (also called continuation algorithms) in **Xyce**. For a more complete description of solver options, see the Xyce Reference Guide [1].

HOMOTOPY Algorithms Available in **Xyce**

There are two general types of homotopy which are available in **Xyce**. The first (which is set with `.options nonlin continuation=1`), is a simple natural parameter homotopy, in which the homotopy parameter is an already-defined input parameter to a device model or instance. This algorithm can be useful, but often is not. The most obvious natural parameters to use (the magnitudes of independent sources) tend to lead to turning points in the continuation.

The second is an algorithm which is designed especially for MOSFET circuits [6]. This algorithm involves two internal MOSFET model parameters, one for the MOSFET gain, and the other for the nonlinearity of the current-voltage relationship. This algorithm is invoked with `.options nonlin continuation=2`. This algorithm has proved to be very effective in large MOSFET circuits.

10.2 Examples

MOSFET Homotopy

Figure 10.1 contains a MOSFET homotopy example netlist. Note that this is a usage example - the circuit itself does not require homotopy to run. Circuits which are complex enough to require homotopy would not fit on a single page. The lines pertainent to the homotopy algorithm are highlighed in red.

Explanation of Parameters, Best Practice

Note that this example shows one set of options, but there are a number of other combinations of options that will work.

```
THIS CIRCUIT IS A MOS LEVEL 1 MODEL CMOS INVERTER
.TRAN 20ns 30us 0 5ns
.PRINT tran v(vout) v(in) v(1)
.options timeint reltol=5e-3 abstol=1e-3
.options linsol ksparse=1


* HOMOTOPY Options
.options device voltlim=0

.options nonlin continuation=2

.options loca stepper=0 predictor=0 stepcontrol=1
+ initialvalue=0.0 minvalue=-1.0 maxvalue=1.0
+ initialstepsize=0.2 minstepsize=1.0e-4
+ maxstepsize=5.0 aggressiveness=1.0
+ maxsteps=100 maxnliters=200


VDDdev  VDD 0 5V
RIN IN 1 1K
VIN1  1 0  5V PULSE (5V 0V 1.5us 5ns 5ns 1.5us 3us)
R1    VOUT  0  10K
C2    VOUT  0  0.1p
MN1   VOUT  IN 0   0   CD4012_NMOS  L=5u W=175u
MP1   VOUT  IN VDD VDD CD4012_PMOS  L=5u W=270u
.MODEL cd4012_pmos PMOS
.MODEL cd4012_nmos NMOS
.END
```

Figure 10.1. Example MOSFET homotopy netlist.

There are a number of "best practice" rules, which are illustrated by the example in figure 10.1. They are:

- `voltlim=0`. This is generally required - the homotopy algorithms will usually break if this is not set.
- `continuation=2`. This specifies that we are using the special MOSFET homotopy. This is a 2-pass homotopy, in which first a parameter having to do with the gain is swept from 0 to 1, and then a parameter relating to the nonlinearity of the transfer curve is swept from 0 to 1.
- `initialvalue=0.0`. This is required.
- `maxvalue=1.0`. This is required.
- `stepcontrol=1`. This specifies that the homotopy steps are adaptive, rather than constant. This is recommended.
- `maxsteps=100`. This sets the maximum number of continuation steps for each parameter. For the special MOSFET continuation (which has 2 parameters), this means a maximum of 200 steps.
- `maxnliters=200`. This is the maximum number of nonlinear iterations, and has precedence over the similar number which can be set on the .options nonlin line.
- `aggressiveness=1.0`. This refers to the step size control algorithm, and the value of this parameter can be anything from 0.0 to 1.0. 1.0 is the most aggressive. In practice, try starting with this set to 1.0. If the solver fails, then reset to a smaller number.

Natural Parameter Homotopy

Figure 10.2 contains a natural parameter homotopy netlist. It is the same circuit as was used in figure 10.1, except that some of the parameters are different. As before, the lines pertainent to the homotopy algorithm are highlighed in red.

Explanation of Parameters, Best Practice

There are a few differences between the netlist in figure 10.1 and figure 10.2. They are:

- `continuation=1`. Sets the algorithm to use natural parameter homotopy.
- `conparam=VDDdev`. If using natural parameter homotopy, this is required. It sets which input parameter to use. The parameter name is subject to the same rules as parameter used by the `.STEP` capability. (See section 9.2). In this case the parameter is the magnitude of the DC voltage source, VDDdev. For this type of voltage source, it was possible to use the default device parameter (see section 9.5)

```
THIS CIRCUIT IS A MOS LEVEL 1 MODEL CMOS INVERTER
.TRAN 20ns 30us 0 5ns
.PRINT tran v(vout) v(in) v(1)
.options timeint reltol=5e-3 abstol=1e-3
.options linsol ksparse=1

* HOMOTOPY Options
.options device voltlim=0

.options nonlin continuation=1

.options loca stepper=0 predictor=0 stepcontrol=1
+ conparam=VDDdev
+ initialvalue=0.0 minvalue=-1.0 maxvalue=5.0
+ initialstepsize=0.2 minstepsize=1.0e-4
+ maxstepsize=5.0 aggressiveness=1.0
+ maxsteps=100 maxnliters=200


VDDdev  VDD 0 5V
RIN IN 1 1K
VIN1  1 0  5V PULSE (5V 0V 1.5us 5ns 5ns 1.5us 3us)
R1    VOUT  0  10K
C2    VOUT  0  0.1p
MN1   VOUT  IN 0   0   CD4012_NMOS  L=5u W=175u
MP1   VOUT  IN VDD VDD CD4012_PMOS  L=5u W=270u
.MODEL cd4012_pmos PMOS
.MODEL cd4012_nmos NMOS
.END
```

Figure 10.2. Example natural parameter homotopy netlist.

Using the magnitudes of independent voltage and current sources is a fairly obvious approach. Unfortunately, it doesn't seem to work very well in practice.

11. Results Output and Evaluation Options

Chapter Overview

This chapter illustrates how to output simulation results to data or output files.

- Section 11.1, *Control of Results Output*
- Section 11.2, *Additional Output Options*
- Section 11.3, *Evaluating Solution Results*

11.1 Control of Results Output

Xyce supports only one solution output command, .PRINT. .PRINT is quite flexible, and supports several output formats.

.PRINT Command

The .PRINT command sends the analysis results to an output file. **Xyce** supports several options on the .PRINT line of netlists that control the format of the output. The syntax for the command is as follows:

- .PRINT <analysis type> [options] <output variable(s)>

Example: .PRINT TRAN FILE=Output.prn V(3) V(2) V(4)

Table 11.1 gives the various options currently available to the .PRINT command. For further information, see the **Xyce** Reference Guide [1].

11.2 Additional Output Options

.OPTIONS OUTPUT Command

The main purpose of the .OPTIONS OUTPUT command is to provide control of the frequency at which data is written to files specified by .PRINT TRAN commands. This can be especially useful in controlling the size of the results file for simulations which required a large number of time steps. An additional benefit is that reducing the output frequency from the default, which outputs results at every time-step, can improve performance. The format for controlling the output frequency is:

- .OPTIONS OUTPUT INITIAL_INTERVAL=<interval> [<t0> <i0> [<t1> <i1> ...]]

where INITIAL_INTERVAL=<interval> specifies the starting interval time for output and <tx ix> specifies later simulation times (tx) where the output interval will change to (ix).

The following example shows the output being requested (via the netlist .OPTIONS OUTPUT command) every .1μs for the first 10μs, every 1μs for the next 10μs, and every 5μs for the remainder of the simulation:

Example: .OPTIONS OUTPUT INITIAL_INTERVAL=.1us 10us 1us 20us 5us

Option...	Action...
FORMAT=<STD|NOINDEX|PROBE>	Controls the output format. The STD format outputs data in standard columns. The NOINDEX format is the same as the standard format except that the index column is omitted. The PROBE format specifies that the output should be formatted to be compatible with the PSpice Probe plotting utility. The *default* is STD.
FILE=<output filename>	Allows the user to specify the output filename. The *default* is the netlist filename with the characters ".prn" appended (e.g., foo.cir.prn where foo.cir was the input netlist filename.
WIDTH=<print field width>	Allows the user to control the column width for the output data.
PRECISION=<floating point precision>	Controls the number of significant digits past the decimal point.
FILTER=<filter floor value>	Specifies the absolute value below which output variables will be printed as 0.0.
DELIMITER=<TAB|COMMA>	Specifies an alternate delimiter between columns of output in the STD output format.

Table 11.1. .PRINT command options.

Note: **Xyce** will output data at the next time that is greater-than or equal to the current interval time. This means that output might not correspond exactly to the time intervals due to the adaptive time stepping algorithm.

11.3 Evaluating Solution Results

This section describes how to view graphical waveform analysis of the simulation results generated by **Xyce**. You can use the solution output features of **Xyce** in conjunction with graphing tools (e.g., TecPlot, gnuplot, MS Excel, etc.) to analyze graphically the waveform data created by a **Xyce** circuit simulation (see Figure 11.1 below for an example plot using TecPlot, `http://www.amtec.com`). In addition, **Xyce** is able to output `.csd` files which can be read by the PSpice Probe utility to view the results. See the PSpice Users Guide [2] for instructions on using the Probe tool.

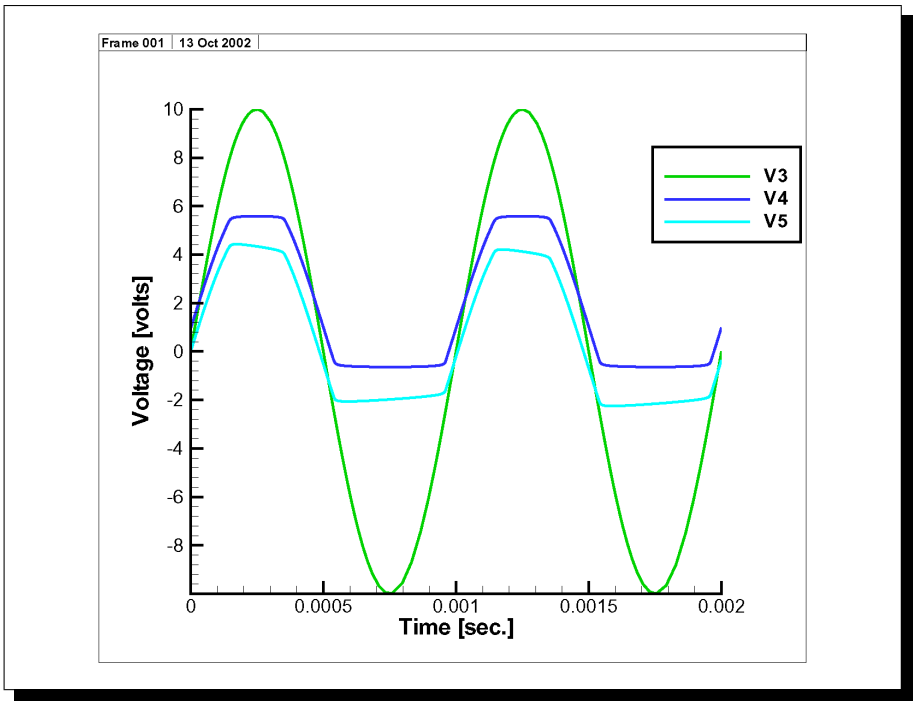

Figure 11.1. TecPlot plot of diode clipper circuit transient response from **Xyce** `.prn` file.

Xyce produces two types of output: the simulation output file and the waveform data file. The calculations and results reported in the simulation output file can be thought of as an audit trail of the simulation. However, graphical analysis of data in the waveform data file is the most useful and accommodating way to evaluate simulation results.

12. Running in Parallel

Chapter Overview

This chapter gives instructions on how to run a parallel version of **Xyce** built with support for the Message Passing Interface (MPI) version of **Xyce** on a supported parallel computer. It includes the following sections:

- Section 12.1, *Simple Parallel Execution Example*
- Section 12.2, *Running Xyce in Parallel*
- Section 12.3, *Partitioning Options*

12.1 Simple Parallel Execution Example

This section demonstrates a simple example of running **Xyce** in parallel environments using an existing netlist. Currently, the user must login locally to the parallel platform and use the command line to start the job.

This example assumes a standard MPI implementation and uses the most common set of associated commands.

1. Make sure a parallel-capable executable (`xmpirun`) for **Xyce** is available. The installation procedure described in Section 1.2 will create this script.
2. Execute **Xyce** using the following command:

```
xmpirun -np <# procs> [options] <netlist_file>
```

[`options`] are the command line arguments for **Xyce** and `mpirun`. See the **Xyce** Reference Guide for details.

For running on most platforms, the procedure outlined above will be sufficient, although, to improve efficiency, the user may use graph partitioning methods, which is the subject of Section 12.3. It is critical that the number of processors used is less than the number of devices and voltage nodes in the netlist.

12.2 Running Xyce in Parallel

A parallel version of **Xyce** is available for several different platforms as shown in Table 6.2. Running **Xyce** in parallel requires that the correct version of `mpirun` is used. Use the script `xmpirun` to call the correct version with the appropriate parameters. For example, to run **Xyce** on two processors with an example netlist, type:

```
xmpirun -np 2 anExampleNetlist.cir
```

In general the number of processors is specified by using the `-np` argument to the appropriate `mpirun` command. Some specific considerations are given below.

Running Xyce under MPICH

The MPICH implementation of MPI requires that there exist a file of machines on which to run. On RedHat Linux this is installed in /usr/lib/mpich/share. On FreeBSD this is installed

in /usr/local/mpich/share. This file must contain one line for each machine on which a process may be started. If you do not have write access to the directory in which the default machines file is stored you may specify an alternate file with the `-machinefile <machinefilename>` option to `mpirun`.

MPICH executes parallel jobs by using the remote shell (rsh) or secure shell (ssh) to the target machine. You may, therefore, be prompted for a password when starting up a multiple processor job.

Running Xyce under LAM MPI

Unlike MPICH, LAM MPI requires a daemon process to be running on each machine that will service parallel jobs. This daemon is started by using the `lamboot` program. By default, `lamboot` will run a daemon on the local machine, but it may be given a file name containing a list of machines for multiple-machine jobs. Consult the `bhost` man page for the format of the file.

`lamboot` runs a program called `lamd` which will remain running until it is halted. As long as `lamd` is running you may continue to run parallel jobs. Halt `lamd` using the `lamhalt` command.

12.3 Partitioning Options

Xyce currently has two graph partitioning options available. These partitioning utilities subdivide the circuit problem into sections that are then distributed to the nodes [1] (processors) on a parallel computer. A good partition can have a dramatic effect on the parallel performance of circuit simulation run. Basically, there are two key components to a good partition: 1) achieving an effective load balance and 2) minimizing communication overhead. An effective load balance ensures that the computational load of the calculation is equally distributed among the available processors. Minimizing communication overhead seeks to distribute the problem in a way that reduces the impact of underlying message passing during the simulation run. **Xyce** has integrated within it two partitioning libraries - the **Chaco** static partitioner and the **ZOLTAN** library of parallel partitioning heuristics.

Chaco Static Partitioning of Circuit

Chaco is accessible using the `'.OPTIONS PARALLEL PARTITIONER=0'` line in the netlist. By adding this line to the netlist, **Chaco** will be used to partition the initial circuit before it is dis-

[1] The term "node" is used here in the parallel computing context to refer to a unit of a parallel computer. This is a more general term the that of "processor"; in many distributed memory computers, two or more processors share a memory unit and are collectively referred to as a "node". Note also that the context should prevent confusion with "circuit nodes".

tributed to processors. **Chaco** partitioning can be controlled through a `'Chaco_User_Params'` file that must be in the execution directory. See the **Chaco User Guide** [7] for details.

Currently, one parameter is available for **Chaco** partitioning: using `'DISTRIBINDSRCNODES=0'` as a parallel option can be very effective for an improved partitioning, especially for large digital circuits. However, limitations are placed on `RESTART` and `OUTPUT` due to the renaming and distribution of some independent sources and their associated voltage nodes. Restarting can only be used for an identical number of processors and partitioning. Current cannot be output for the distributed voltage sources and voltage cannot be output for the distributed voltage nodes.

Zoltan Partitioning of Linear System

Zoltan is accessible through the `'.OPTIONS LINSOL'` control line in the netlist. By adding an options `'TR_LOADBALANCE=1'` to the linear options, the linear system is statically load balanced based on the graph of the Jacobian matrix. The local system is also reordered based on nested dissection which should improve conditioning and minimize fill. These techniques can be very effective for improving the efficiency of the iterative linear solvers.

Recommended Partitioning and Solver Options

A recommended set of options for parallel problems includes singleton filtering as well as Chaco and Zoltan partitioning of the circuit and linear system respectively. The additional settings can improve the performance of the preconditioned linear solver.

- `.OPTIONS PARALLEL PARTITIONER=0 DISTRIBINDSRCNODES=0`
- `.OPTIONS LINSOL TR_LOADBALANCE=1 TR_SINGLETON_FILTER=1 TR_SOLVERMAP=1 TR_REINDEX=1 USE_IFPACK_PRECOND=1`

Bibliography

[1] Scott A. Hutchinson, Eric R. Keiter, Robert J. Hoekstra, Lon J. Waters, Thomas V. Russo, Eric L. Rankin, Roger P. Pawlowski, and Steven D. Wix. Xyce parallel electronic simulator: Reference guide, version 2.0. Technical Report SAND2003-xxxx, Sandia National Laboratories, Albuquerque, NM, December 2003.

[2] Orcad PSpice User's Guide. Technical report, Orcad, Inc., 1998.

[3] M. S. Eldred, A. A. Giunta, B. G. van Bloemen Waanders, S. F. Wojtkiewicz Jr., W. E. Hart, and M. P. Alleva. DAKOTA, A multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis. Version 3.0 Reference Manual. Technical Report SAND2001-3796, Sandia National Laboratories, Albuquerque, NM, April 2002.

[4] A. S. Grove. *Physics and Technology of Semiconductor Devices*. John Wiley and Sons, Inc., 1967.

[5] H. A. Watts, E. R. Keiter, S. A. Hutchinson, and R. J. Hoekstra. Time integration for the Xyce parallel electronic simulator. In *ISCAS 01*, October 2000.

[6] J. Roychowdhury. *Private Communication*, 2003.

[7] Bruce Hendrickson and Robert Leland. The Chaco User's Guide: Version 2.0. Technical Report SAND94–2692, Sandia National Laboratories, Albuquerque, NM, December 1994.

Index

DISTRIBUTION:

1 Steven P. Castillo
Klipsch School of Electrical and Computer Engineering
New Mexico State University
Box 3-o
Las Cruces, NM 88003

1 Kwong T. Ng
Klipsch School of Electrical and Computer Engineering
New Mexico State University
Box 3-o
Las Cruces, NM 88003

1 Nick Hitchon
Electrical and Computer Engineering
University of Wisconsin
1415 Engineering Drive
Madison, WI 53706

1 Mark Kushner
Department of Electrical and Computer Engineering
University of Illinois
1406 W. Green Street
Urbana, IL 61801

1 Andrew J. Christlieb
Department of Mathematics
University of Michigan
2470 East Hall
Ann Arbor, MI 48109

1 Ron Kielkowski
RCG Research, Inc
8605 Allisonville Rd, Suite 370
Indianapolis, In 46250

1 Mike Davis
Software Federation, Inc.
211 Highview Drive
Boulder, Co 80304

1 Wendland Beezhold
Idaho Accelerator Center
1500 Alvin Ricken Drive
Pocatello, Idaho 83201

1 Kartikeya Mayaram
Department of Electrical and Computer Engineering
Oregon State University
Corvallis, OR 97331-3211

1 Linda Petzold
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106-5070

1 Jaijeet Roychowdhury
4-174 EE/CSci Building
200 Union Street S.E.
University of Minnesota
Minneapolis, MN 55455

1 C.-J. Richard Shi
VLSI and Electronic Design Automation
210 EE/CSE Bldg.
Box 352500
University of Washington
Seattle, WA 98195

1 Homer F. Walker
WPI Mathematical Sciences
100 Institute Road
Worcester, MA 01609

1 Dan Yergeau
CISX 334
Via Ortega
Stanford, CA 94305-4075

1 Masha Sosonkina
319 Heller Hall
10 University Dr.
Duluth, MN 55812

1 Misha Elena Kilmer
113 Bromfield-Pearson Bldg.
Tufts University
Medford, MA 02155

1 Tim Davis
P.O. Box 116120
University of Florida
Gainesville, FL 32611-6120

1 Achim Basermann
C&C Research Laboratories, NEC Europe Ltd.
Rathausallee 10
D-53757 Sankt Augustin
Germany

1 Philip A. Wilsey
Experimental Computing Laboratory
Department of Electrical & Computer Engineering and Computer Science
College of Engineering
P.O. Box 210030
University of Cincinnati
Cincinnati, Ohio 45221-0030

1 Dale E. Martin
Clifton Labs
3678 Fawnrun Dr.
Cincinnati, OH 45241

1 MS 0151
Tom Hunter, 09000

1 MS 0513
Al Romig, 01000

1 MS 0457
John Stichman, 02000

1 MS 0321
Bill Camp, 09200

1 MS 0841
Thomas C. Bickel, 09100

1 MS 1079
Marion Scott, 01700

1 MS 9003
Kenneth E. Washington, 08900

1 MS 0318
Paul Yarrington, 09230

1 MS 1071
Mike Knoll, 01730

1 MS 0310
Robert Leland, 09220

1 MS 0316
Sudip Dosanjh, 09233

1 MS 0525
Paul V. Plunkett, 01734

1 MS 0835
J. Michael McGlaun, 09140

1 MS 0835
Steven N. Kempka, 09141

1 MS 0826
John D. Zepper, 09143

1 MS 0824
Jaime L. Moya, 09130

1 MS 0828
Martin Pilch, 09133

1 MS 0139
Stephen E. Lott, 09905

1 MS 0310
Mark D. Rintoul, 09212

1 MS 1110
David Womble, 09214

1 MS 1111
Bruce Hendrickson, 09215

1 MS 1110
Neil Pundit, 09223

1 MS 1110
Doug Doerfler, 09224

1 MS 0822
Philip Heermann, 09227

1 MS 0819
Edward Boucheron, 09231

1 MS 0820
Patrick Chavez, 09232

1 MS 0316
John Aidun, 09235

10 MS 0316
Scott A. Hutchinson, 09233

1 MS 0316
Eric R. Keiter, 09233

1 MS 0316
Robert J. Hoekstra, 09233

1 MS 0316
Joseph P. Castro, 09233

1 MS 0316
David R. Gardner, 09233

1 MS 0316
Gary Hennigan, 09233

1 MS 0316
Roger Pawlowski, 09233

1 MS 0316
Richard Schiek, 09233

1 MS 1111
John N. Shadid, 09233

1 MS 1111
Andrew Salinger, 09233

1 MS 0847
Scott Mitchell, 09211

1 MS 0847
Mike Eldred, 09211

1 MS 0847
Tim Trucano, 09211

1 MS 0847
Bart van Bloemen Waanders, 09211

1 MS 0196
Elebeoba May, 09212

1 MS 1110
Todd Coffey, 09214

1 MS 1110
David Day, 09214

1 MS 1110
Mike Heroux, 09214

1 MS 1110
James Willenbring, 09214

1 MS 1111
Karen Devine, 09215

1 MS 0310
Jim Ang, 09220

1 MS 1109
Robert Benner, 09224

1 MS 0822
Pat Crossno, 09227

1 MS 0822
David Rogers, 09227

1 MS 0316
Harry Hjalmarson, 09235

1 MS 0525
Steven D. Wix, 01734

1 MS 0525
Thomas V. Russo, 01734

1 MS 0525
Lon Waters, 01734

1 MS 0525
Regina Schells, 01734

1 MS 0525
Carolyn Bogdan, 01734

1 MS 0525
Mike Deveney, 01734

1 MS 0525
Raymond B. Heath, 01734

1 MS 0525
Ronald Sikorksi, 01734

1 MS 0525
Albert Nunez, 01734

1 MS 1081
Paul E. Dodd, 01762

1 MS 0660
Roger F. Billau, 09519

1 MS 0874
Robert Brocato, 01751

1 MS 1081
Charles E. Hembree, 01739

1 MS 0311
Greg Lyons, 02616

1 MS 0311
Martin Stevenson, 02616

1 MS 0328
Fred Anderson, 02612

1 MS 0537
Perry Molley, 02331

1 MS 0537
Siviengxay Limary, 02331

1 MS 0537
John Dye, 02331

1 MS 0537
Barbara Wampler, 02331

1 MS 0537
Doug Weiss, 02333

1 MS 0537
Scott Holswade, 02333

1 MS 0481
Joel Brown, 02132

1 MS 0405
Todd R. Jones, 12333

1 MS 0405
Thomas D. Brown, 12333

1 MS 0405
Donald C. Evans, 12333

1 MS 9101
Rex Eastin, 08232

1 MS 9101
Seung Choi, 08235

1 MS 9409
William P. Ballard, 08730

1 MS 9202
Kathryn R. Hughes, 08205

1 MS 9202
Rene L. Bierbaum, 08205

1 MS 9202
Kenneth D. Marx, 08205

1 MS 9202
Stephen L. Brandon, 08205

1 MS 9217
Stephen W. Thomas, 08950

1 MS 9217
Tamara G. Kolda, 08950

1 MS 9217
Kevin R. Long, 08950

1 MS 1179
Leonard Lorence, 15341

1 MS 1179
David E. Beutler, 15341

1 MS 1179
Brian Franke, 15341

1 MS 0835
Randy Lorber, 09141

1 MS 1152
Mark L. Kiefer, 01642

1 MS 9018
Central Technical Files, 8945-1

2 MS 0899
Technical Library, 9616

1 MS 0612
Review & Approval Desk, for DOE/OSTI, 9612